CENTRUL DE PREGĂTIRE PENTRU PERFORMANȚĂ HAI LA OLIMPIADĂ!
DISCIPLINA **INFORMATICĂ**
JUDEȚUL **SUCEAVA**

Titlul lecției: **Tablouri unidimensionale: sortări**

Data: 26.11.2016

Profesor: Petrică Galan

Grupa a IX-a Seniori, **locul de desfășurare:** Colegiul Național „Ștefan cel Mare”, Suceava

Cuprins

I. Sortare - generalități

Sortarea tablourilor unidimensionale (vectori) reprezintă rearanjarea elementelor acestora astfel încât între valorile lor există o relație de ordine. (crescător/descrescător, A-Z/Z-A, etc.).

1. Sortare prin selecție directă;
2. Sortare prin selecția minimului/maximului;
3. Sortare prin metoda bulelor (BubbleSort);
4. Sortare prin inserție;
5. Sortare prin determinarea poziției prin numărare;
6. Sortare rapidă (QuickSort).

Algoritmii următori ordonează vectorii crescător.

II. Algoritmi de sortare a vectorilor

1. Selecție directă

Această metodă presupune aducerea pe primul loc a celui mai mic element dintre cele n elemente ale tabloului pe prima poziție, apoi aducerea celui mai mic element dintre cele n-1 ramase pe cea de a doua poziție ș.a.
Această „aducere” se realizează prin comparări și interschimbări succesive a valorii curente cu cele ce o urmează în vector.
Complexitatea algoritmului este n^2.
	`int aux;`
	`for(int i=0;i<n-1;i++)`
	`  for(int j=i+1;j<n;j++)`
	`  if(a[j]<a[i])`
	`  {`
	`  aux=a[i];`
	`  a[i]=a[j];`
	`  a[j]=aux;`
	`  }`

2. Sortare prin selecția minimului/maximului;

Această metodă presupune interschimbarea valorii curente cu valoarea cea mai mica dintre toate elementele vectorului începând cu cea curentă.
Exemplu: elementul curent este primul din tablou. Pornind de la acesta se caută valoarea minimă din tablou și apoi se interschimbă cu cea curentă, după care algoritmul se reia plecând de la al doilea element. Procedeul continuă până la penultimul element.
Complexitatea algoritmului este n^2.
	`int aux, Min, pozMin;`
	`for(int i=0;i<n;i++)`
	`{`
	`  Min=a[i], pozMin=i;`
	`  for(int j=i+1;j<n;j++)`
	`  if(a[j]<Min)`
	`  {`
	`  Min=a[j];`
	`  pozMin=j;`
	`  }`
	`  aux=a[i];`
	`  a[i]=a[pozMin];`
	`  a[pozMin]=aux;`
	`}`

3. Sortare prin metoda bulelor (BubbleSort);

Această metodă presupune parcurgerea tabloului în mod repetat și compararea fiecărui element din tablou cu succesorul său. Dacă nu sunt în ordine cele două elemente se interschimbă.
Dacă la o parcurgere se face măcar o interschimbare, vectorul se parcurge din nou.
Vectorul este sortat dacă **la o parcurgere nu se face nici o interschimbare**.
Complexitatea algoritmului în cazul general este n^2. În cazul cel mai favorabil (vectorul este ordonat crescător) complexitatea este liniară (n).

```
int gata=0, aux;
while(!gata)
{
    gata=1;
    for(int i=0;i<n-1;i++)
        if(a[i]>a[i+1])
        {
            aux=a[i];
            a[i]=a[i+1];
            a[i+1]=aux;
            gata=0;
        }
}
```

4. Sortare prin inserţie;

Începând cu al doilea element al tabloului, fiecare element este inserat pe poziția adecvată în subtabloul care îl precedă.
Această metodă presupune inserarea elementului curent din vector pe o poziție astfel încât elementele să rămână ordonate pană la poziția curentă, astfel: elementul curent este extras din vector, apoi este identificată poziția pe care ar trebui așezat astfel încât ordinea până la elementul curent să fie păstrată, după care se face efectiv inserția.
Inserția elementelor se poate face și într-un al doilea vector, ale cărui elemente vor fi copiate la final în vectorul inițial.
Complexitatea algoritmului în cazul general este n^2. În cazul cel mai favorabil (vectorul este ordonat crescător) complexitatea este liniară (n).

```
int aux, j;
for(int i=1;i<n;i++)
{
    aux=a[i];
    j=i-1;
    while(j>=0 && a[j]>aux)
    {
        a[j+1]=a[j];
        j--;
    }
    a[j+1]=aux;
}
```

5. Sortare prin determinarea poziției prin numărare;

Această metodă presupune numărarea pentru fiecare element al tabloului, a numărului de valori mai mici decât acesta într-un vector de apariții (ap). Această informație este apoi utilizată pentru a determina poziția fiecărui element în tabloul ordonat. Daca sunt cinci elemente mai mici decât x în tablou, atunci în tabloul ordonat x se va afla pe poziția a șasea.
Pentru așezarea elementelor în ordinea dată de vectorul de apariții se utilizează un al treilea vector (dest).
La final se copiază elementele din vectorul auxiliar (dest) în cel inițial.

```
int i, j, dest[n], ap[n];
for(int i=0;i<n;i++)
    ap[i]=0;
for(int i=0;i<n-1;i++)
    for(int j=i+1;j<n;j++)
        if(a[i]>a[j])
            ap[i]++;
        else
            ap[j]++;
for(int i=0;i<n;i++)
    dest[ap[i]]=a[i];
for(int i=0;i<n;i++)
    a[i]=dest[i];
```

6. Sortare rapidă (QuickSort)

Sortarea rapidă este o metodă de sortare eficientă ce se bazează pe ideea alegerii unui element denumit pivot și așezării acestuia în cadrul vectorului astfel încât toate elementele mai mici decât acesta să se găsească în stânga sa iar cele mai mari în dreapta.
În acest mod pivotul se va afla pe poziția finală în cadrul vectorului ordonat.
Algoritmul se reia pentru elementele din stânga și din dreapta pivotului.
Alegerea pivotului influențează performanța algoritmului în mare măsură.
Complexitatea algoritmului în cazul general este $n*\log_2 n$ iar în cazul cel mai defavorabil n^2.

```
int a[100];
int pozitie(int st,int dr)
{
    int aux,stare=0;
    while(st<dr)
    {
        if(a[st]>a[dr])
        {
            aux=a[st];
            a[st]=a[dr];
            a[dr]=aux;
            stare=!stare;
        }
        if(stare==0)
            dr--;
        else
            st++;
    }
    return st;
}
void qsort(int st,int dr)
{
    int p;
    if(st<dr)
    {
        p=pozitie(st,dr);
        qsort(st,p-1);
        qsort(p+1,dr);
    }
}
```

III. Probleme propuse

#1608 Sortare Divizori

Cerința
Se dau n numere naturale nenule. Ordonați descrescător cele n numere după numărul lor de divizori.
Date de intrare
Fișierul de intrare sortare_divizori.in conține pe prima linie numărul n, iar pe a doua linie n numere naturale nenule separate prin câte un spațiu.
Date de ieșire
Fișierul de ieșire sortare_divizori.out va conține cele n numere aflate pe a doua linie a fișierului de intrare ordonate descrescător după numărul de divizori.
Restricții și precizări
$1 \le n \le 1000$
numerele de pe a doua linie a fișierului de intrare vor fi mai mici decât 1.000.000.000
dacă există mai multe numere care au același număr de divizori, acestea vor fi ordonate crescător
Exemplu
sortare_divizori.in
5
12 20 4 100 13
sortare_divizori.out
100 12 20 4 13
Explicație
12 are 6 divizori, 20 are 6 divizori, 4 are 3 divizori, 100 are 9 divizori, 13 are 2 divizori, 12 și 20 au același număr de divizori. Așadar ordinea va fi 100 12 20 4 13.

#511 KSort

Cerința

Se dă un vector cu n elemente, numere naturale și un număr k. Ordonați crescător primele k elemente ale vectorului și descrescător ultimele n-k elemente.

Date de intrare

Programul citește de la tastatură numerele n și k, iar apoi n numere naturale, reprezentând elementele vectorului.

Date de ieșire

Programul va afișa pe ecran elementele vectorului, separate prin exact un spațiu, după efectuarea operațiilor cerute.

Restricții și precizări

$1 \leq k < n \leq 1000$

cele n numere citite vor fi mai mici decât 1.000.000.000

Exemplu

Intrare

7 3

13 1 10 15 3 7 11

Ieșire

1 10 13 15 11 7 3

#185 CifInit

Cerința

Se citește de la tastatură un număr natural n, apoi n numere naturale. Să se afișeze cel mai mic număr care poate fi scris folosind prima cifră a numerelor citite.

Date de intrare

Programul citește de la tastatură numărul n, iar apoi cele n numere naturale, separate prin spatii.

Date de ieșire

Programul afișează pe ecran numărul MIN, cel mai mic număr care poate fi scris folosind prima cifră a numerelor citite.

Restricții și precizări

$0 < n < 1000$

cele n numere citite vor fi nenule și mai mici decât 1.000.000.000

Exemplu

Intrare

5

100 312 276 985 5021

Ieșire

12359

#164 HalfSort2

Se dă un vector cu n elemente numere întregi, n fiind număr par.

Cerința

Să se ordoneze crescător elementele situate pe poziții pare în vector și descrescător elementele situate pe poziții impare.

Date de intrare

Fișierul de intrare halfsort2.in conține pe prima linie numărul n si pe a doua linie n numere întregi separate prin spatii.

Date de ieșire

Fișierul de ieșire halfsort2.out va conține pe prima linie cele n elemente ale vectorului, ordonate conform cerinței, separate printr-un spațiu.

Restricții și precizări

$0 < n \leq 100$

valoarea absolută a numerelor de pe a doua linie a fișierului de intrare va fi mai mică decât 2^{30}

indicii elementelor vectorului sunt 1,2,...,n

Exemplu

halfsort2.in

6

8 9 9 4 5 7

halfsort2.out

9 4 8 7 5 9

#183 sortPP

Cerința

Să se ordoneze crescător elementele pătrat perfect ale unui șir dat, fără a afecta elementele care nu sunt pătrat perfect.

Date de intrare

Programul citește de la tastatură numărul n, iar apoi n numere naturale, separate prin spatii, reprezentând elementele vectorului.

Date de ieșire

Programul afișează pe ecran, separate prin spatii, cele n elemente ale vectorului, după sortare.

Restricții și precizări

$1 \leq n \leq 1000$

cele n numere citite vor fi mai mici decât 50000

Exemplu

Intrare

8

9 15 16 4 5 1 7 9

Ieșire

1 15 4 9 5 9 7 16

#273 OrdSume

Cerința

Se dă un șir cu n elemente, numere naturale. Să se afișeze, în ordine crescătoare, toate valorile distincte care se pot obține ca sumă de două valori distincte din șir.

Date de intrare

Fișierul de intrare ordsume.in conține pe prima linie numărul n, iar pe a doua cele n elemente ale șirului dat, separate prin spatii.

Date de ieșire

Fișierul de ieșire ordsume.out va conține pe prima linie, în ordine crescătoare, toate valorile distincte care se pot obține ca sumă de două valori distincte din șir, separate printr-un spațiu.

Restricții și precizări

$1 \leq n \leq 100$

numerele de pe a doua linie a fișierului de intrare vor avea cel mult 8 cifre

Exemplu

ordsume.in

4

1 7 3 5

ordsume.out

4 6 8 10 12